# DEVELOPING THE COMPILER TO UPGRADE FEATURES OF OBJECT ORIENTED PROGRAMMING (OO) C++

[1]Abdul Razaque [2]Nyembo. Salama [3]Abhilasha Tibrewal [4]Enam ul Haq [5]Khaled Elliethy
[1]arazaque@bridgeport.edu [2]nsalama@bridgeport.edu [3]abhilash@bridgeport.edu [4]ehaque@bridgeport.edu elleithy@bridgeport.edu

**Department of Computer Science and Engineering**
**University of Bridgeport, USA**

## Abstract:

The latest deployment of emerging technologies with advances in internet and web enabled services change the style of conducting business and managing the daily life. The latest and faster technology has made the world as global village for exchanging the information and service delivery. However, developing the new software and interactive applications for these environments perpetuate to be more challenging and complex issue. To handle these complex tasks, many programming languages are developed to meet the requirements for all of the applications but flaws in programming languages are still continuing. With design and development of Object-Oriented Programming (OOP) languages, handling the complexity issues associated with the design, development, maintaining the software and applications, have been controlled with some extend but not fully guaranteed goals are achieved. OOP provides the solution and satisfaction in form of encapsulation, inheritance, polymorphism, Class/Object and data abstraction. Despite of provision of such rich features, OOP has some minor flaws, which causes the weak performance. **T**his paper introduces and handles the flaws of parsing related with C++ language. The major focus of the contribution is to improve an efficiency of parsing associated with current C++ parsers; such as delayed loop statements, nested multi-line Comments, friend functions boolean expression. To handle these issues, we introduce Java Compiler Compiler (Java-CC), which maintains the Lexical and the Parser. Java CC also provides the facility to write flexible coding and definitions in OOP C++. Second, we implement new concept of Java-CC and get the desired results. Finally, we conclude the paper and give the future directions to make C++ more richer language.

**Keywords:** Parser, lexer, C++,Delayed Loop Statements, expressing Boolean expressions, Friend Function and Nested Multi-line comments.

## 1. Introduction

The beauty of any programming language depends on enhanced features, which make the language very efficient. The performance increasing features are data size, code size, execution speed and memory footprint at the run time and space utilized by compiler/link process/editor [1].Some Programming languages are designed for performing the specific tasks(domain-specific modeling) such as Maxima and Mathematica for symbols of mathematics, Sequential Query Language (SQL) for relational database queries, Generic Eclipse Modeling System for making the diagramming languages, R and S languages for statistics. From other side, second major category is general-purpose programming languages, such as Perl ,Java,, C, C++, and third category is of general purpose modeling design language such as Unified Modeling Language (UML). From earliest days,The improvements in an efficiency

of programming languages have been a major focusing design goals. The designer also tried to integrate the zero overhead rule for those features, which are not used in a program [1].On the basis of this principle, C++ has been designed with more extra rich features such as Object-Oriented features including encapsulation in classes, use of constructors, destructors. Inheritance, Polymorphism, which provide run-time selection, Generic programming such as functions, template classes, member functions, Exception handling, Garbage collection, Persistence, Standard library and Missing [3]. Despite of all these healthier features, C++ faces some minor issues, which cause the lack of performance and compatibility. Our contribution highlights those issues and provides the solutions to make the C++ more versatile language. Modifications to C++ language in widespread is not easy task and even slightly changes require linguistic discussion [2]. we present here limitations of C++ such as Delayed Loop Statements, expressing Boolean expressions as digits, Friend Function and Nested Multi-line comments. To this conclusion, we deem that this paper makes reasonable contribution. The rest of the paper is organized as follows. In Section II, We highlight the limitations with proposed implementation and finally conclude the paper.

## 2. Methodology and Proposed Implementation

To tackle the weak points of C++ language, we introduce the new idea with integration of Lexer and parser supported with Java compiler (JAVACC). This new concept of combining the features of programming and parser code gives the functionality of compiler and introduces the novel way of solving the problem.

### 2.1. Boolean Expressions

If the Boolean expressions are defined in the form of non-boolean expressions; The existing parsers do not allow to use digits in Boolean area. To tackle this issue, we use our compiler to solve the problem. We give simple an example

while(1){...} // while contains non Boolean expression

for (int x =0; 1; x++) {...} // for contains non_Boolean expression

**To handle the above weakness; the following sample code is the best solution**

(<IDENTIFIER> | | Arithmetic Operation ()) Boolean Operation()

(<IDENTIFIER> | <DIGIT> |

<FLOAT> | < DOUBLED>) ( ( <LOGAND> |

<LOGOR>

(<IDENTIFIER>

| Arithmetic Operation ()) Boolean Operation()

(<IDENTIFIER> | <DIGIT> | <FLOAT> | < DOUBLED>)

### 2.2 Delayed Loop Statements

They causes the endless loops, To resolve this issue, we need to set clock concept to delay the program .The following simple program shows the delayed loop statement.

for(int i=0;i<10;i++);//Delayed Loop

//Statement

while(x<y); //Delayed Loop Statement

To handle this issue, we provide the solution,

which makes the C++ more richer.

<FOR> <LPARAN>

(<SIGNED> | <UNSIGNED> )? (Data types() |

<REGISTER> |

<EXTERN>) <IDENTIFIER>

(<EQUAL> <DIGIT>)? ("," <IDENTIFIER>

(<EQUAL> <DIGIT>)? )*

<SEMICOLON>

(Boolean Expression() |

Boolean Single())<SEMICOLON>

(<IDENTIFIER><PLUSPLUS> |

<PLUSPLUS><IDENTIFIER> |

<IDENTIFIER><MINUSMINUS> |

<MINUSMINUS><IDENTIFIER>)

<RPARAN>

<SEMICOLON>

### 2.3. Friend Functions

They are non-member function of other classes, which access the protected and private members of a class within the scope of the class but not out side of the class (main function). non-member function are declared inside the class as friends by using the keyword friend. If the friends functions are declared in order to access the protected and public data members and member functions but why they do not have access to protected and private date members in main function. To resolve this weakness, our designed front end part of the compiler, the lexer and the parser, showed promising results to fulfill the objectives by providing a complete solution to the Friend

```
class Object : private class1,private class2
{
private:
        int y;
protected:
        int y;
Public:
public:
// member functions
void  func3(){}

Object(){z=4;}
void func1(){}
friend void get_function(Product s)
{
s->x;
t.n;
s.func1();
s->func2(g);
int a,b;
cout <<j<<endl;
} friend int get_function(Product t){
t->m;
void operator<<(Student y){}
~Object(){}
};
```

Friend Functions are Detected

Variable s accesses x which could be private variable

Variable t accesses n which could be private variable

Variable s accesses f1 which could be private variable

Variable s accesses f2 which could be private variable

Variable t accesses m which could be private variable

## 2.4. Nested Multi-line comments

The compiler of C++ does not have capability to resolve the Nested Multi-Line Comments; technically it does not look big issue but causes the lack of features of language. We handle this issue by SKIP that is part of JAVACC. "SKIP" does the process of matching the regular expressions ignored by taken Manager. Through this processes, the special tokens are passed to the parser to trigger them with real neighboring tokens by using the special token field in token class. These tokens are useful to process lexical entities such as Nested Multi-Line comments.

For example:

```
 /*    This    is    multi_line
comment */
 /*    This    is    multi_line
comment */
 /* This is nested multi_line comment */
/* This is nested multi_line comment */
```

An existing Parsers do not determine the Nested Multi-line comments but same task is done with our parser with following solution.

**TOKEN_MGR_DECLS**:

```
{
public static int   single line comments   = 0;
public static int   multilinecomments=0;
public static int counter=0;
<DEFAULT> SKIP;
{
        <"/*">
        {
        counter++;
        Switch To (INCOMMEN T);
        }
```

```
                    }
<INCOMMENTT> SKIP;
 {
        <"*/">{multilinecomments++;counter
        --;

        if(counter==0)
          {
        Switch To (DEFAULT);
        }  }
```

## Conclusion:

In this paper, we have discussed the weakness of object oriented Programming Language C++ such as delayed loop statements, nested multi-line comments, friends functions and Boolean expressions, which affects the beauty of language. To resolve this issue, we have developed compiler, which integrates the features of laxer and parser. To achieve the objectives, we have provided the best solutions with deployment of developed compiler. This small effort can motivate the people to utilize their best efforts to improve programming languages by removing the minor bugs and make the programming languages more flexible and feature-oriented as much as possible. In future, we will discuss the important features of polymorphisms, constructor and destructor in classes.


## Acknowledgement

Thanks to Professor Abhilasha Tibrewal who provided the valuable support through her best teaching method. Due to her expertise in C++ made this small but meaningful effort possible and thanks to Professor Khaled Elliethy who advised to take the course C++. We are also grateful to Semantic and distributed computing network (SDCN) group, Mohammad Ali Jinnah University, Islamabad Pakistan who helped in implementing the ideas of integrating the lexer and parser to handle the weakness of C++ language.


## References:


[1].Dave Abrahams, Mike Ball et al., "Technical Report on C++ Performance", ISO/IEC PDTR 18015, August 11, 2003.

[2]. *Bjarne Stroustrup*,"C and C++: Case Studies in Compatibility", Part of a three-article series from "The C/C++ Users Journal", AT&T Labs, July-September 2002.

[3]. Douglas C. Schmidt," An Overview of C++",